1990-09

Design and implementation of a Primal Simplex Network Optimizer in C

Solveson, Keith D.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF A PRIMAL SIMPLEX NETWORK OPTIMIZER IN C

by

Keith D. Solveson

September, 1990

Thesis Advisor: Gordon H. Bradley

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution/Availability of Report: Approved for public release; distribution is unlimited.	
2b Declassification/Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*): OR	7a Name of Monitoring Organization: Naval Postgraduate School	
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000		7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	
8a Name of Funding/Sponsoring Organization	8b Office Symbol (*if applicable*)	9 Procurement Instrument Identification Number	
8c Address (*city, state, and ZIP code*)		10 Source of Funding Numbers	
		Program Element No	Project No / Task No / Work Unit Accession No

11 Title (*include security classification*): DESIGN AND IMPLEMENTATION OF A PRIMAL SIMPLEX NETWORK OPTIMIZER IN C

12 Personal Author(s): Keith D. Solveson

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	September 1990	61

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	network, C, primal, transportation, transshipment, simplex

19 Abstract (*continue on reverse if necessary and identify by block number*)

This thesis documents the design and implementation of an efficient primal simplex capacitated transshipment network optimizer, SNET, written in the C programming language. It describes a general symbolic network algorithm, discusses fundamental decisions regarding data structures and essential functions and their relationship to the network algorithm, and then details SNET's development. Development tools used in this project, including standard test problems, profilers, timing routines, external drivers, and debuggers, are also covered.

The resulting solver, SNET, is quite fast on standard NETGEN test problems, approximately twice as fast as a primal simplex network solver written in FORTRAN. The effect of tuning parameters on SNET's performance is minimal.

20 Distribution/Availability of Abstract: ☒ unclassified/unlimited ☐ same as report ☐ DTIC users	21 Abstract Security Classification: Unclassified	
22a Name of Responsible Individual: Gordon H. Bradley	22b Telephone (*include Area code*): (408) 646-2359	22c Office Symbol: OR/BZ

DD FORM 1473,84 MAR — 83 APR edition may be used until exhausted; All other editions are obsolete

Approved for public release; distribution is unlimited.

Design and Implementation of a Primal
Simplex Network Optimizer in C

by

Keith D. Solveson
Captain, United States Army
B.S., United States Military Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1990

Author: [redacted]

Keith D. Solveson

Approved by: [redacted]

Gordon H. Bradley, Thesis Advisor

[redacted]

Roger Stemp, Second Reader

[redacted]

Peter Purdue, Chairman,
Department of Operations Research

ABSTRACT

This thesis documents the design and implementation of an efficient primal simplex capacitated transshipment network optimizer, SNET, written in the C programming language. It describes a general symbolic network algorithm, discusses fundamental decisions regarding data structures and essential functions and their relationship to the network algorithm, and then details SNET's development. Development tools used in this project, including standard test problems, profilers, timing routines, external drivers, and debuggers, are also covered.

The resulting solver, SNET, is quite fast on standard NETGEN test problems, approximately twice as fast as a primal simplex network solver written in FORTRAN. The effect of tuning parameters on SNET's performance is minimal.

THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

For their noteworthy advice (and considerable patience), I would like to thank the following individuals for assisting in my thesis research:

- My Advisor. Professor Gordon H. Bradley, for our extensive discussions on all facets of networks; but specifically, our discussions on graph theory for general network problems and the theoretical and practical implications of my design decisions. A great deal of my learning during this thesis arose from these discussions.
- My Second Reader. LCDR Roger Stemp, for direction during my initial forays as a novice C programmer, and for supporting the concept of a fast network solver written in standard, portable, ANSI C.
- LCDR John Yurchak, for his advice on the efficient use of the C programming language and help with complex C problems.
- Professor Richard Rosenthal, for access to his unpublished notes on symbolic methods of solving the capacitated transshipment network problem. They were clear and concise, unlike many other publications that I encountered.
- My Wife. Denise, for her support and unending enthusiasm.

I. INTRODUCTION

A minimum cost capacitated transshipment (min-cost) network can model many important problems that industry and the military must solve repeatedly: shipment of commodities, assignment of personnel and resources to tasks and requirements, routing of vehicles, and design of distribution and communications systems, to name but a few.

This report describes the development of SNET, a fast and efficient C program for solving minimum cost capacitated network problems, which the author wrote for his thesis research. It describes min-cost network models, documents key decisions, explains choices of data structures and information representations, and describes in detail the critical functions needed to implement the network optimization algorithm. The development tools that significantly aided in the design and implementation of SNET are also discussed, as is the effect of tuning parameters. Performance tests comparing GNET and SNET for speed are included.

The source code for SNET is not included with this report. To obtain a copy of SNET, please contact the author's thesis advisor: Professor Gordon H. Bradley.

A. NETWORK PROBLEMS

Minimum cost capacitated transshipment networks, hereafter referred to as *networks*, are a special class of linear programming (LP) problems. They can be used to model problems where

- Each variable can be interpreted as the (integer) amount of commodity flow on a conduit or arc.
- Each constraint can be interpreted as a point or node that interacts with commodity flow arcs.
- Each node may either supply commodity to the network, consume commodity from the network, or transfer commodity to another node.
- Each arc is connected to two network nodes.
- The total supply of commodity into the network equals the total consumption of commodity from the network.

Examples of problems that meet these criteria abound in operational, strategic and planning arenas. Min-cost networks are frequently used to model the following general problems, which are common to both the military and industrial world:

- Communication Networks

- Personnel Assignments
- Movement of Units, Supplies, and Commodities
- Logistic / Production Planning
- Financial Planning

Additionally, there are many specialized models unique to the military community:

- Wartime Allocation of Aggregated Assets
- Specialized Weapon to Target Assignments
- Manpower Mobilization
- Distribution of Intelligence Collection Assets

From these examples, it is clear that network models are not only important, but can also be quite large. For example, a military manpower mobilization model to determine U.S. Army wartime officer assignments could have over one hundred thousand nodes and one million arcs. Thus, in solving network problems, the speed and efficiency of the solver are important considerations.

B. NETWORK FORMULATIONS AND SOLVERS

The network matrix formulation (Figure 1) is the same as the general LP matrix formulation. However, the network constraint matrix has a special property: its entries can only be +1, -1 or 0, and each column must have exactly one +1 and one -1.

A network can also be interpreted as a directed graph. Each node in the graph is either exogenous, supplying flow into the network or demanding flow from it, or endogenous, transfering (or transshipping) flow through the network. Within the network total supply must equal total demand. Each node acts as a constraint, as the sum of flow into and out of each node must equal zero. Each arc acts as a variable with a lower and upper flow capacity bound and a linear cost proportional to its current rate of flow. Network vocabulary often uses the terms arc and variable interchangeably. An optimal network solution transfers the flow through the network at the minimum possible cost. More detailed information on networks can be obtained from many standard linear programming references. [Ref. 1: pp. 404-439]

This paper concentrates on an implementation of the primal simplex algorithm, but efficient solvers exist for many network algorithms. GNET [Ref. 2] and RNET [Ref. 3] are sophisticated primal simplex programs. KILTER [Ref. 4] is a fast primal-dual implementation and RELAXT-II [Ref. 5] uses the relaxation method. Each of them has contributed to the advance of practical network solvers.

```
Minimize     cx ,
Subject to   Ax = b ,
             l <= x <= u .

Where  x is the decision variable vector
       c is the cost vector
       A is the constraint coefficient matrix
       b is the right hand side constraint vector
       l and u are the variable bound vectors
```

Figure 1. Network Linear Program Matrix Formulation.

C. TYPOGRAPHIC CONVENTIONS

In this paper, C programming language functions and variable names will be displayed in lower case boldface type. This convention accurately represents the actual variable and function names as the C language is case sensitive. *Important terminology* will be highlighted in italics.

II. NETWORK ALGORITHMS

SNET is based on the symbolic algorithm, a specialized version of the bounded variable simplex method. The symbolic algorithm solves min-cost network problems using a graph interpretation of the model instead of the standard simplex tableau. Though most standard LP texts derive the bounded variable simplex method, many do not cover the symbolic algorithm. This chapter fills that void. It summarizes the critical information needed by the bounded variable simplex method, and then presents the symbolic algorithm, an alternative method of manipulating that information for networks.

A. THE BOUNDED VARIABLE SIMPLEX ALGORITHM

The tableau from the bounded variable simplex method contains four essential vectors of variables: basic variables, nonbasic variables at their lower bound, nonbasic variables at their upper bound, and the current values of the basic variables. With these four vectors and the information needed to begin the simplex method (the constraint matrix and cost vector), one can recreate the rest of the tableau; dual prices, reduced costs, and the value of the objective function; and hence, any basic feasible solution. [Ref. 1: pp. 201-212]

When the LP under consideration is a network, recreating the feasible solution is simplified. First, by convention, all exogenous network variables and arc costs are integer valued. If this convention is not natural to the problem, it is achieved by scaling. Second, since the entries in the constraint matrix of a network consist of only positive ones, negative ones, or zeros, the basis inversion and subsequent matrix multiplications to recreate dual prices and reduced costs are all integer operations [Ref. 6: pp. 305-306].

The symbolic algorithm provides an easier way to recreate the feasible solution. The Basis-Tree theorem, discovered by Koopmans and Hitchcock [Ref. 7], states that:

> *A set of columns of the constraint matrix comprise a basis if and only if the corresponding set of arcs form a spanning tree.*

Thus, the collection of basic arcs (variables) may be interpreted as a spanning tree. One can associate with each basic arc its cost and current flow. If the nonbasic arcs and their costs are maintained in two lists, one for arcs at their upper bound and the other for arcs

at their lower bound, all the information needed to recreate the feasible solution is present.

After an initial basic feasible solution is obtained, the bounded variable simplex method has four major steps:

1. Calculate the dual price of each constraint and the reduced cost of each variable.
2. Select a favorable variable to enter the basis. If none exists, then the solution is optimal.
3. Determine which variable should leave the basis. If none would leave the basis, then the solution is unbounded.
4. Adjust relevant variables and update the basis.

The combination of steps 1 and 2 is referred to as *selecting candidates.* Steps 3 and 4, together, are referred to as *pivoting the basis.* The symbolic algorithm uses the same basic steps, but its calculations emanate from the graph representation of the spanning tree and not from matrix manipulation.

B. THE SYMBOLIC ALGORITHM

To obtain an initial feasible basis (spanning tree) for the symbolic algorithm, artificial arcs connect each node to the root node. The root node is the only artificial node in the spanning tree. It corresponds to the redundant row, equal to minus the sum of the other rows, that can be appended to the constraint matrix. This row ensures that each artificial variable will each have exactly one +1 and one -1 in their column. Recall that a pure network problem has exactly one redundant constraint, hence its basis would be singular and noninvertable. Therefore any feasible basis must contain at least one artificial arc, though it may have zero flow.

Artificial arcs for supply nodes flow from the supply node to the root node. Similarly, arcs for sink nodes flow from the root node to the sink nodes. Artificial arcs for endogenous nodes may flow either way. Initially, all flow passes through the root. The cost for each artificial arc should be high, to drive it out of the basis. An artificial arc's capacity can be set equal to its initial flow, so no increase in its flow is possible.

1. Dual Prices and Reduced Costs

Dual-prices (DP) are uniquely associated with each constraint in a general LP and, hence, with each node in a network. In the symbolic algorithm, a node's DP can be interpreted as the cost that a unit of flow would incur while travelling from the node itself to any arbitrary node. The root node can serve as that arbitrary node. Therefore, a node's DP can be equal to the total cost that a unit of flow would incur as it travels

from the node to the root node. If an arc flows from the current node to the root node, then its cost is added to the DP. If the arc flows away from the root and towards the current node, then its cost is subtracted from the DP.

The reduced cost of each basic arc is zero, as in every variant of the simplex method. To obtain the reduced cost of nonbasic arcs simply add the arc's cost to the DP of the arc's head and subtract the DP of the arc's tail.

2. Entering Variable

In a min-cost problem, a favorable variable entering the basis has the potential to reduce the value of the objective function. An arc entering the basis at its lower bound with a negative reduced cost will decrease the objective function as its flow increases. Similarly, an arc entering the basis at its upper bound with a positive reduced cost will decrease the objective function as its flow decreases. The greater the magnitude of the reduced cost, the more favorable the arc. Therefore, arcs may be ranked or sorted by their reduced costs.

At each pivot, the reduction of the objective function's value is equal to the entering arc's flow change, which may be zero, multiplied by its reduced cost. Although choosing the entering variable with the most favorable reduced cost will not insure the largest improvement in the objective function, extensive experimentation over many years has shown that, on average, it is best to choose entering variables this way.

As before, if no favorable arcs can be found, the solution is optimal.

3. Exiting Variable

Selecting an arc to leave the basis is the next step in the symbolic algorithm. Recall that the basis arcs form a spanning tree. When the arc entering the basis is added to this tree, a unique cycle is formed [Ref. 8: pp. 32-33]. As flow is adjusted around this cycle, the feasible solution approaches optimality. Each of the cycle arcs must be oriented either with or against the incoming arc. If the incoming arc enters the basis at its lower bound, then its flow can only increase. If flow increases on the incoming arc, it will increase on arcs oriented with the incoming arc and decrease on arcs oriented against it. If the incoming arc enters the basis at its upper bound, then its flow can only decrease. If flow decreases on the incoming arc, it will decrease on arcs oriented with the incoming arc and increase on arcs oriented against it. The arc which limits the change in flow induced by the incoming arc will be the exiting variable.

Flow can be limited in one of three ways. First, the incoming arc itself could reach its opposite bound. For example, if the incoming arc enters the basis at its lower bound, the flow around the cycle can change until the incoming arc reaches its upper

bound. Second, flow on an arc already in the basis can increase until that arc reaches its upper bound. Third, flow on an arc already in the basis can decrease until that arc reaches its lower bound. All flow changes are integer valued and it is quite possible that multiple arcs will limit flow simultaneously. In this case, any (one) flow limiting arc in the cycle may be chosen to leave the basis.

As in other simplex methods, if no arc reaches a limit the solution is unbounded; however, in practical problems this is unlikely to occur.

4. **Flow Adjustment**

Cycle flow can be adjusted after the limiting arc has been identified. During flow adjustment, the limiting arc's flow will be driven to either its lower or upper bound. The arc is then removed from the basis and placed in the proper nonbasic list. This step completes the symbolic network algorithm.

III. FUNDAMENTAL DESIGN DECISIONS

During SNET's conceptual phase, several fundamental design decisions were made: design objectives, choice of a programming language, and composition of essential data structures. These fundamental decisions impacted upon SNET's development in two ways.

First, they provided a common philosophical foundation. This foundation assures that the program, taken as a whole, is easier to understand and maintain than a collection of less similar modules. A common design philosophy also eases the programmer's task by providing a standard frame of reference for each module. Easing the programmer's efforts leads to fewer coding errors and reduces development time.

Second, the fundamental design decisions restricted the possible alternatives that could be employed. The choice of a language naturally limits one to the features available in that language. Likewise, the information present in the initial data structures may be insufficient for effective implementation of an alternative unforeseen during the conceptual phase.

These factors, guidance and restriction, dictated that careful consideration be given to the fundamental design decisions.

A. DESIGN OBJECTIVES

A number of desirable design objectives were identified during the conceptual phase:

- Solve large scale problems
- Simple
- Easily Understood
- Modular
- Portable
- Fast

Although the network formulation itself is not inherently large, the problems that networks represent often are. Thus, for a network solver to be applicable to a wide range of problems, it must be able to handle large scale networks. How large is large scale? The scale or size of a problem is defined by the number of nodes and arcs in the network. In SNET, *large* is the smaller of two limits which depend upon the host computer: integer capability and available memory.

Integer capability requires that the sum of the number of arcs and nodes in a network be less than the maximum integer that the host can represent. Typical maximum integers are $2^{(16-1)}$ (32,768) for a personal computer (PC) and $2^{(32-1)}$ (2,147,483,648) for a workstation or mainframe computer. For example, the integer capability of a PC would allow it to handle any network with less than 2^{15} nodes and arcs.

The amount of host memory, both core and virtual, available for dynamic data representation can also limit problem size. Each node's dynamic representation requires a fixed amount of memory, as does each arc. A certain amount of memory is also required for program overhead. The sum of these three memory requirements (node, arc, and overhead) cannot exceed the amount of data memory available. For example, if a node requires 40 bytes, an arc requires 20 bytes, 4K of overhead is present, and 64K of host memory is available, then any combination of nodes and arcs that could be stored in less than 60K would be allowable. A problem with 200 arcs and 1400 nodes would be acceptable, but one with 300 arcs and 1351 nodes would not.

The smaller of these two limits, integer capability and host memory, determines the maximum problem size. Thus, in the example given, host memory available is the limiting factor.

How simple and easily understood a design is depends upon one's education and experience. SNET can be understood by individuals with moderate experience in network linear programming and a minimal knowledge of the C programming language. One graduate course in each should suffice.

Simple and easily understood also implies that complex strategies will be employed only when there is a significant gain in performance. Thus a complicated scheme, that offered only modest improvement over a basic one, would not be utilized.

Simple designs are, by definition, easily explained. Thus, even though source code may not be immediately legible to the uninitiated, one should be able to explain it with a minimum of sophisticated verbiage. This increases the odds that others may be able to contribute to an improved solver in the future. Furthermore, software maintenance costs can be drastically reduced.

By keeping solver design simple, one can concentrate on principles of the network algorithm, rather than on the mechanics of the solver itself. The author believes that this philosophy offers the best chance for future improvements in algorithm development.

A modular design encapsulates frequently used or functionally unique portions of code into subroutines which are called without reference to their internal design. It also

allows different algorithms to be incorporated into the program without major restructuring. One merely exchanges the new algorithm module for the old module without concern for possible global effects. Thus modular code is easy to modify and allows rapid development and testing of new algorithms.

There is, of course, a cost for the benefits of modular code. If the overhead of entering and exiting the module is substantial, considerable time could be wasted by the modular structure. In this case, one would have to carefully balance the advantages of modular structure against its cost.

A portable design can be easily transferred among many different host computers. A single portable program can, with minimal recoding, serve in several environments. The savings in programming effort are obvious. The disadvantages are not. If a programmer uses only features that are completely portable, then often he cannot employ advanced, nonstandard features of a language that are more efficient for a particular computer. The author employs only portable features.

Finally, the benefit of a fast program is obvious: problems can be solved in less time.

B. PROGRAMMING LANGUAGE

SNET is written in the C programming language in accordance with the proposed ANSI C standard. Optimizing a network is an intense computational problem. For years FORTRAN was the dominant language for such tasks. However, within the last ten years, C has gained prominence [Ref. 9: p. 152] for intense computational problems in many communities since it has several advantages over FORTRAN.

First, C generally executes faster than FORTRAN. Although more noticeable on UNIX machines, whose operating systems are written in C, this is generally true for most computers. Comparing assembly language generated from C to that generated from FORTRAN, reveals that C frequently produces shorter and more efficient code segments [Ref. 9: pp. 155-156].

Second, C users can employ true pointer (memory address) variables. FORTRAN users cannot. The components of a network, nodes and arcs, are not independent. Rather, they are coupled together in a determinate fashion. For such problems, pointers are a natural method of linking components together. Although one can implement array based pseudo-pointers in any language, the use of true pointers is preferable. A simple example illustrates why.

Suppose one wishes to find the child of a node. The pointer based construct for this operation is

answer = node -> child ,

whereas the array based construct is

answer = child(node) .

In the pointer based construct, node is a pointer to the address of a structure (a collection of data variables) and child is a member of that structure. The member, child, is always located at a constant offset from the beginning of the node structure. Given the location of the structure, the program immediately knows the location of each member, and need only go to that location and return the value of the variable located there. The process of going to a location and returning the value at that location is known as *indirection*. Thus, retrieving the answer, which is a pointer in this case, requires one indirection.

The array based construct requires significantly more effort. In this construct, node is an integer valued variable and child is an integer array. The computer must first retrieve the value of node, then using that value and the size of the array elements, calculate the offset from the beginning of the array. It must then add that offset to the array's address to return the required address and, finally, perform an indirection on that address. Thus, retrieving the answer, which is an integer for the array based construct, requires several steps in addition to the indirection.

Third, C allows structures, collections of (possibly dissimilar) data variables, to be easily implemented. FORTRAN has no structure facilities. Structures, sometimes referred to as *records* in other languages, allow related groups of variables to be treated in a natural manner. This intuitive method of organizing complicated data simplifies and clarifies programming requirements by allowing these groups to be handled as a unit. For networks, whose nodes and arcs each have several different types and items of information, structures greatly simplify data retrieval and manipulation.

Fourth, the C language protocol specifies standard libraries of functions for input/output, type conversion, math, memory management, and similar operations. These functions are, technically, not part of the C language. Rather, they provide a support environment within which C can efficiently work. Other languages, including FORTRAN, incorporate these functions into the language itself. In C, each compiler, which is usually machine specific, has its own set of function libraries. Since these library interfaces are standardized, the means of accessing each function from within C is always the same, regardless of the computer used. But as the libraries are designed for

each machine, the function's implementation can be locally optimized. Thus, the C function libraries support development of portable programs which will run efficiently on most computers.

Fifth, C can enter and exit its function modules with minimal overhead. Thus modular design, breaking large, complex tasks into smaller, simpler ones, can be used without excessive fear of inefficiency in C.

C. DATA STRUCTURES

SNET's network data is maintained in two data structures: node and arc. A third structure, the arc list element (ALE), is used when pivoting the basis. The C code needed to implement these three structures is in Figure 2.

Both nodes and arcs have properties and values that are intrinsic to them. All variables are integer valued. The maximum value that a variable may assume determines if it should be typed as an int (integer) or long (long integer) variable.

Each node has a label that identifies its external association point. It may have flow that is exogenous (external) to the network. Within the basis tree, each node has a specific depth and a dual price that is relative to the root node. The depth and dual price are likely to change as the basis is updated.

An arc also has several important elements. It has a cost per unit flow, a maximum capacity, and often a minimum capacity. Of course, it will always have a current flow value, which may be zero, and a reduced cost. The latter two quantities may be frequently updated.

However, as noted before, nodes and arcs do not exist in a vacuum. They are coupled together in a determinate fashion. SNET uses pointers to link components together. Note that pointers point *to* other structures, not *from* them.

In SNET, each node can be connected to three other nodes: a parent node, a sibling node, and a child node. Thus the node structure contains three node pointers: parent (p), sibling (s), and child (c). Recall that each node in the network, except the artificial root node, always has exactly one parent node and is connected to that parent node by exactly one arc. Thus the node structure contains one arc pointer: parent arc (pa). Each node is always accounted for in the basis spanning tree, so there is no need to track it elsewhere. It is also important to note that each node contains sufficient information to enter the basis tree and trace completely through it. Therefore, these four pointers are sufficient to link the node into the basis tree at all times.

```
/* Structures */
   struct nodetype
   {  int               lbl,     /* label, i.e. node nbr    */
                        d ;      /* tree depth              */
      long              b,       /* exogenous flow          */
                        dp ;     /* dual price              */
      struct nodetype   *p,      /* parent                  */
                        *s,      /* sibling                 */
                        *c ;     /* child                   */
      struct arctype    *pa ;    /* parent arc              */
   } ;
   typedef struct nodetype node ;

   struct arctype
   {  long              l,       /* lower capacity          */
                        u,       /* upper capacity          */
                        c,       /* unit flow cost          */
                        r,       /* reduced cost            */
                        x ;      /* flow                    */
      struct nodetype   *h,      /* head node               */
                        *t ;     /* tail node               */
      struct arctype    *next ;  /* next arc in 'out' list  */
   } ;
   typedef struct arctype arc ;

   struct arclistelemtype
   {  struct arclistelemtype  *next ;
      struct arctype          *a ;
   } ;

   typedef struct arclistelemtype ale ;
```

Figure 2. SNET Data Structures.

Arcs are simpler to handle. When in the basis, each arc is associated with one head node and one tail node. When out of the basis, the arc will be located in one of three linked lists of similar arcs, depending on whether its flow is at its lower or upper bound, or the arc is being considered as a candidate for entering the basis. Thus the arc structure contains two node pointers, head (h) and tail (t), and one arc pointer, next arc (next).

It is informative to note that array based languages must also track the interlinking relationships and integer values just detailed. Generally, a set of parallel arrays is the only practical method for non-pointer languages to handle this information. As noted earlier in this chapter, parallel array operations are inefficient compared to pointer operations. Programs employing parallel arrays suffer a significant performance degradation.

The third structure, the ALE, identifies the arcs in the unique cycle formed during pivoting. It has two pointers: next ALE (next), and arc (a). Cycle arcs are pointed to by linked lists of ALEs, where next links elements of the list together, and a points to the cycle arc that that element identifies.

Another possible implementation to efficiently mark the cycle involves placing an additional arc pointer in the arc structure. However, this design must place that additional pointer in each arc structure; significantly increasing the memory required to represent the dynamic data structures.

IV. PROGRAM DEVELOPMENT

The symbolic algorithm and fundamental design decision form the structure upon which SNET is built. But the construction of an efficient program requires many other important elements:

- Symbolic and other Constants
- Global Variables
- Key Functions: Collecting candidates, selecting a candidate, pivoting
- External Files

This chapter describes those elements. It also covers tools used to test and tune the program.

A. SYMBOLIC AND OTHER CONSTANTS

Several constants are important for the symbolic algorithm itself, and for the C implementation of the algorithm. SNET uses two types of constants: symbolic and program defined.

The numeric values of symbolic constants are set prior to program compilation and are determined by the scale of expected problems and capabilities of the host computer. These constants are called *symbolic* because they can be represented by a symbol or word (usually denoted by capital letters in C). SNET has four symbolic constants: MAXNODES, MAXARCS, INFINITY, and MAXCAN.

MAXNODES and MAXARCS are determined primarily by the scale of the expected problems. They dimension arrays and control looping structures. MAXNODES must be at least one greater than the number of real nodes in the network to accommodate the root node. MAXARCS must be at least one greater than the number of nodes plus the number of arcs in the network. Nodes are included in the count, as each node must initially be connected to the root node by an artificial arc. The maximum value of MAXNODES and MAXARCS is constrained only by the memory capacity of the host computer and its integer range.

INFINITY should equal the largest long integer that the host can represent. It is used to prevent integer overflow and to initialize comparison variables when determining maximum and minimum values among sets.

MAXCAN serves as the upper bound for maxnbrcan, the maximum number of candidates. In empirical studies with problems as large as 35000 arcs, 20000 has been effective as a bound.

Program defined constants are set by the program at run time based upon input data. SNET has two critical program defined constants: Big M (bigm) and maximum number of candidates (maxnbrcan).

Big M is the cost assigned to artificial arcs. It must be large enough to drive their flow to zero if the problem can be solved. In SNET, Big M is one more than half the sum of all positive real arc costs. Since any flow on an artificial arc, either to or from the root node, must also be carried on another artificial arc, the cost assigned by Big M is sufficient to drive their flow to zero.

If the value of Big M is too large for the host computer to represent, SNET halts and so informs the user. In this case, the user may either rescale arc costs or set Big M to a smaller value. However, when a lessor value is chosen and the final solution has positive flow on one or more artificial arcs, it cannot be readily determined whether the problem is infeasible or the value of Big M is too small.

maxnbrcan, (sic) is, in full, the maximum number of favorable candidates encountered and considered for inclusion in the candidate queue during each *collection*. A collection is a single attempt to gather favorable candidates.

maxnbrcan is initially set to MAXCAN. However, this default value may be overridden by the command line argument, percan, which is the percentage of out of basis candidates to be examined. In this case, maxnbrcan is set equal to percan multiplied by the number of real arcs less the number of nodes. The number of nodes is subtracted from the number of arcs before multiplication because each node (except the root node) requires one arc to connect it to the basis tree.

B. GLOBAL VARIABLES

Global variables may (potentially) be accessed anywhere in the program. Local variables are only available within their defining function. Thus, local variables cannot conflict with identically named local variables in different functions. But global variables can cause such conflict. Hence the use of global variables should be restricted to those few variables that are required by several functions.

SNET has eight global variables:

- outhi, outlo, and canque are arc pointers to the heads of three linked arc lists: specifically, arcs out of the basis at their upper bound, arcs out of the basis at their

lower bound, and arcs in the reduced cost candidate queue. They provide the only access to these lists and are used frequently throughout the program.

- arcs and nodes are arrays of pointers to all arcs and all nodes. They are used to set up the initial feasible solution, and to output the final (optimal) feasible solution.
- root is a node pointer to the root node. It is used to establish the initial feasible solution and as a default node to enter the basis tree.
- input and output are FILE pointers to the input and output files.

C. KEY FUNCTIONS

Practical implementations of the symbolic algorithm often organize candidate selection and basis pivoting into three steps:

- Collecting a large group of favorable arcs into a candidate queue
- Getting candidates from the queue until it is empty
- Pivoting candidates, one at a time, into the basis

SNET's principal functions, collect_can, getcan, and pivot, implement these three steps. This section will describe them in sufficient detail for the reader to understand the general implementation of the symbolic algorithm and its relationship to the data structures discussed thus far. Specific C code will not be discussed.

1. Collecting Candidates

Collecting and organizing candidates is accomplished by two functions: collect_can and merge. collect_can (sic) performs a *collection* by (usually) traversing every arc in the out of basis lists. As each arc is visited, collect_can calculates its reduced cost. If the reduced cost is favorable, it determines if a more favorable arc with the same head node has already been traversed. If a more favorable arc has not been traversed, the arc and its predecessor in the linked list are recorded. Thus, after the traversal is complete, the most favorable arc flowing into each node has been recorded. These favorable arcs are then removed from their out of basis list and placed into another linked list, the candidate queue. The merge function [Ref. 10: pp. 113-114] can be used to sort linked lists. collect_can uses merge and a modified binomial comb [Ref. 10: p. 264] to sort the candidate arcs by the absolute value of their reduced cost. After sorting, collect_can sets the canque pointer to the first arc in the candidate queue and returns the number of candidates in the queue.

Unless instructed otherwise, collect_can will traverse the entire out of basis list. However, as discussed earlier, the user can set maxnbrcan, the maximum number of fa-

vorable candidates to examine for inclusion in the candidate queue during each collection.

In summary, collect_can examines the out of basis lists, and builds the candidate queue, a sorted linked list containing the most favorable candidate arc flowing into each node.

2. Picking a Candidate

Candidates must be selected to be pivoted into the basis until the candidate queue is exhausted. getcan (sic), which performs this function, begins at the head of the candidate queue, indicated by canque, and traverses it until a favorable candidate is found or the end of the list is encountered.

Recall that the candidate queue is a sorted linked list of arcs that were very favorable when collect_can was invoked. However, after one or more pivots, they may not remain favorable. Thus, as each arc in the queue is traversed, getcan recalculates its reduced cost. If the reduced cost is still favorable, getcan returns a pointer to that arc and resets canque to the next arc in the list. If the reduced cost is not favorable, getcan returns that arc to the appropriate out of basis list. If the candidate queue is empty, getcan returns a NULL pointer to the calling routine, indicating exhaustion.

3. Pivoting the Basis

Pivoting the basis involves three major steps: selecting a variable to exit the basis, adjusting the relevant variables, and updating the basis. The arc which limits the change in flow induced by the incoming arc, as discussed in Chapter 2, will be the exiting variable. The relevant variables are the arcs on the unique cycle that the incoming arc forms. As flow is adjusted on them, the feasible solution approaches optimality. Since the basis arcs form a spanning tree, updating the basis can be equated to rehanging the basis tree.

Three functions are required to pivot the basis: the principal function, pivot, and two subordinate functions, mature, and calc_ddp.

pivot (sic) first traces out the unique cycle formed by the incoming arc, newarc. As it traces the cycle, pivot must record two items of information on each cycle arc: orientation and location. An arc's orientation is measured relative to newarc's. An arc can either flow *with*, in the same direction as, or *against*, in the opposite direction to, newarc. pivot records orientation by using two ALE lists: a with flow (wflow) list and an against flow (aflow) list. Location is also measured relative to newarc. A cycle arc must be located either on newarc's head side, or newarc's tail side. An arc's location is stored in its next field.

pivot begins the cycle trace by receiving newarc, the incoming arc that getcan selected, from the main program. newarc's (sic) head and tail nodes are then identified. If both nodes are not at the same depth in the spanning tree, pivot travels up the deeper side, noting each arc's location and orientati . until it reaches the node at the same depth as the end node of the untraversed bran . pivot then travels up both sides of the cycle until it reaches the first common node. Again, each arc's data is noted. When the common (joining) node has been reached, each arc in the cycle has been identified and classified with respect to orientation and location.

If newarc was out of the basis at its lower bound, then those arcs oriented in the same direction as newarc can only increase their flow, and those opposing newarc can only decrease their flow. Conversely, if newarc was out of the basis at its upper bound, those arcs oriented in the same direction as newarc can only decrease their flow, and those opposing newarc can only increase their flow. pivot makes this association and renames wflow and aflow to increase and decrease as appropriate. It then traverses both lists, searching each for the arcs allowing the minimum possible change. After locating these arcs, pivot is ready to select the exiting arc, oldarc.

oldarc (sic), will be the arc that limits the flow change induced by newarc. pivot compares the flow limits imposed by newarc itself, the minimum increasing arc and the minimum decreasing arc. The arc with the smallest limit becomes the exiting arc. If there is a tie in limits, pivot selects newarc if possible, otherwise it selects the decreasing arc. The variable delta is set to the limiting quantity of the limiting arc. If delta is greater than zero, the increase and decrease lists are traversed and their flows adjusted appropriately, as is newarc's flow.

If the incoming arc, newarc, is not also the outgoing arc, oldarc, then the basis tree must be rehung. Rehanging first adds newarc to, and removes oldarc from, the spanning tree structure. Next, the *stem*, which consists of the cycle arcs and nodes between newarc and oldarc inclusive, must be adjusted. The mature function standardizes the stem by making each stem node the first child of its parent. This permits adjustment of the stem's parent, child, sibling, and parent arc relationships in an efficient manner.

Next the disposition of oldarc must be resolved. Recall that there are two types of arcs in the symbolic algorithm formulation, real and artificial. A real arc is one that corresponds to an existing arc in the network. An artificial arc does not correspond to any arc in the network. It is used to establish an initial feasible solution in the symbolic network formulation. If oldarc is a real arc, or an artificial arc with positive flow, it will be added to the appropriate out of basis list. Otherwise, it is discarded.

pivot's final step is to call calc_ddp which recalculates the depth and dual price of each rehung node.

D. EXTERNAL (INPUT/OUTPUT) FILES

SNET requires only two I/O files. The input file contains network data in standard network format. The first line of the input file indicates the number of nodes (N) in the problem. All subsequent lines are four-tuples, indicating tail, head, cost, and capacity of the arcs. Exogenous flow, either to or from a node, is indicated by exogenous arcs in the input file. A node's supply is indicated by an arc coming from the source node (node number N+1) to that node. A node's demand is indicated by an arc going from that node to a sink node (node number N+2). The cost of these exogenous arcs is zero and their capacity is the amount of the exogenous flow.

The input filename is normally designated from the command line. If the filename is given on the command line, SNET will explicitly request one. If the input file does not exist or the input data is not in the correct format, SNET will so inform the user and then halt.

The solution is written to the output file. The output file contains the tail, head, flow and cost of each active (nonzero flow) arc and the value of the objective function. Its filename consists of the input filename with *.ans* concatenated to it.

E. TESTING/TUNING TOOLS

Many tools are available to assist a programmer in efficient algorithm implementation. Among the more valuable tools used in SNET's development were debuggers, profilers, data snapshots, and timing routines.

1. Debugger

A debugger is a software package--distinct from the software being developed--that can precisely control and display the execution of compiled source code. It can assist a programmer in locating and correcting program errors (commonly called bugs). Often, more time is spent locating and correcting program errors in a program than was spent writing it. Debuggers, by expediting the correction process, help reduce debugging time and speed the development process.

Several useful facilities are available in most debuggers. Perhaps the most valuable debugging function is the ability to observe and change variable values during program execution. A debugger can display values at any point in time or continuously during execution. It can also allow the user to change the value of a variable, regardless of its previous program defined value.

Different methods of execution control are usually available. A debugger can usually step through code one line or executable statement at a time. Most debuggers offer the option of stepping into or over subordinate functions at the user's discretion. Breakpoints or execution halt points are also a common feature. When the user sets a breakpoint in the source code, the debugger will run the program up to that point, and then return execution control to the user. A few debuggers can also step through code in reverse order, that is, they can run the program *backwards*, allowing decisions and data manipulations to be undone. This facility allows the user to perform complex multiway decision tests with ease.

Although competent debuggers offer many additional features, these two, display and control, were sufficient to locate many obscure errors in the early SNET code. The author highly recommends the use of a debugger for even moderately complex programs.

2. Profiler

A profiler measures a program's performance and execution time. It, like the debugger, is also external to the source code under development. Profiler software tabulates program execution time, either by function or line; counts how many times a line is executed; and tracks how many times a function is called and by whom. It can also monitor activities external to the program, such as CPU interrupts and disk accesses. By monitoring critical activities and providing detailed reports on them, a profiler highlights inefficient program segments and can assist the programmer in refining his code for efficient execution.

Without profilers, programmers have to resort to ad hoc timing and counting functions that must be inserted into the source code and modified as the timing interests change. The information they return is neither as complete nor accurate as that provided by the profiler. Ad hoc functions can also skew the information gathered by the execution resources that they consume.

SNET's tuning process used a profiler that gathered statistics by function. An example profile of an early version of SNET for the net43 problem is given in Figure 3.

In this example, the fscanf function, which reads in the problem, consumes approximately half of the program's time. This time cannot be reduced if the decision to program in standard C is followed. Thus, the programmer must search elsewhere for improvement. The conflict between collect_can and pivot is of interest. In past primal simplex implementations, costing out variables was more expensive than pivoting them

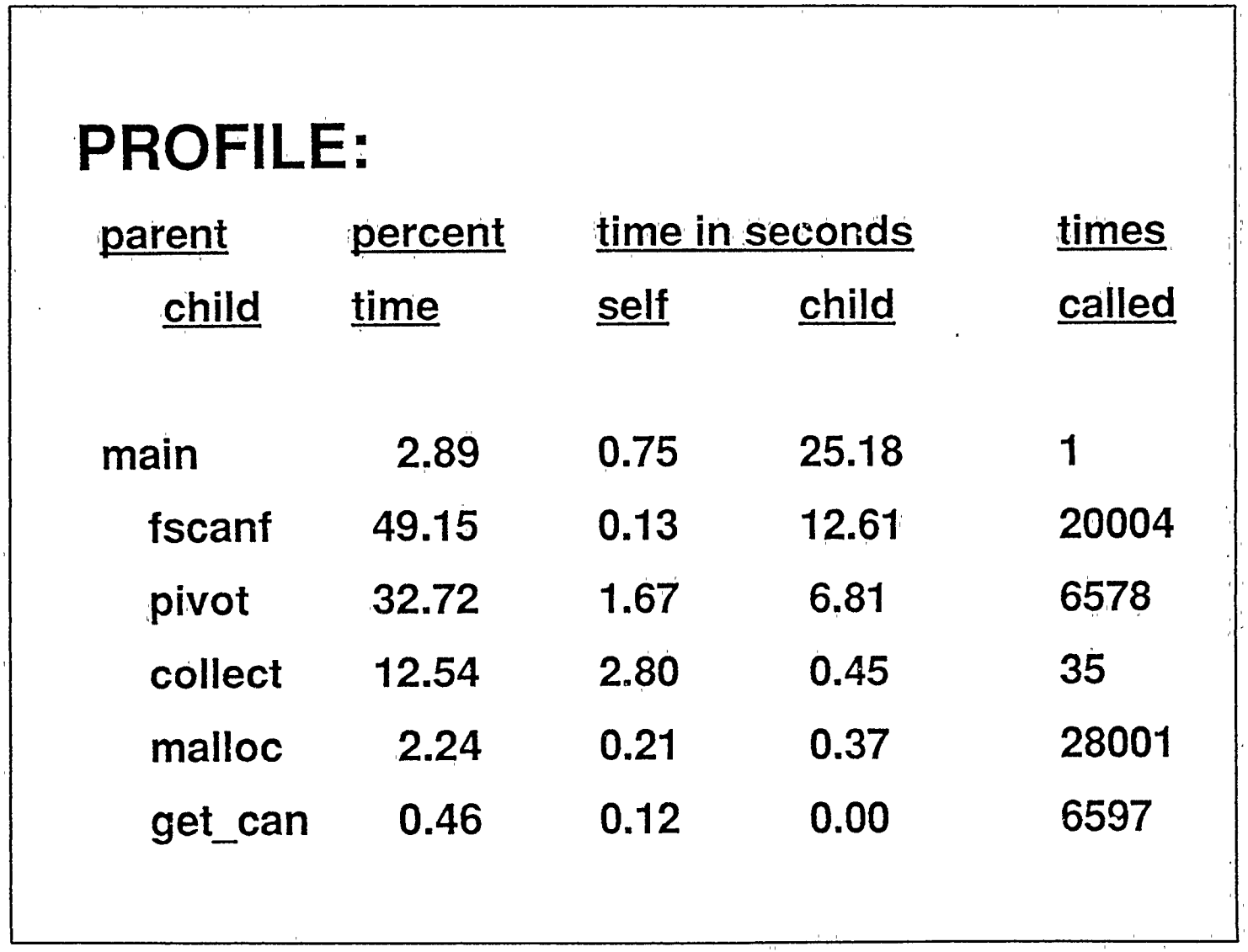


Figure 3. SNET Profile, NETGEN Problem Number 43.

into the basis. As the profile indicates, this is not the case here. Thus, the programmer may wish to examine these two functions in more detail.

3. Data Snapshots

A debugger enables the programmer to view critical variable values during program execution. However, sometimes this is still inadequate to correctly diagnose a program error. In this case, a partial or complete *snapshot* or dump of data structures is required. The three functions, **dumpnode, dumparc,** and **dumpale,** shown in Figure 4, can provide a complete or partial snapshots of the data structures at any point in the program.

A call to these functions with the relevant pointer will direct a snapshot to an external snapshot or dump file. Each function starts at the element indicated by its incoming pointer, and then recursively visits the remaining elements in the relevant structure. **dumpnode** (sic) can be used to examine the basis tree. The out of basis lists are detailed by the **dumparc** function. The **dumpale** function can be used to examine the candidate queue and the cycle identified during basis pivoting. These functions were most useful in diagnosing the rehanging of the basis tree.

4. Timing Routines

In evaluating alternative implementation strategies, it is easy to be misled if one examines strategies on only a few problems. A better approach is to compare the performance of alternative strategies over a wide variety of problems.

External timing routines can easily compare different programs and record their performance over a wide variety of problems. TESTEXEC, a timing routine used in SNET's development, is shown in Appendix C. The programmer can implement the competing strategies in different versions of SNET and then run TESTEXEC to compare the programs. TESTEXEC uses the 40 standard NETGEN [Ref. 11] problems to compare program performance.

Thus, when comparing alternative strategies, one should test them over a wide variety of problems. As TESTEXEC demonstrates, the power of modern computers and programming languages allows a hypothesis to be examined over a large sample set. This can only increase the strength of one's conclusion.

F. TESTING FOR CORRECTNESS

SNET's solutions were tested for correctness by comparing optimal objective function values to those calculated by GNET. GNET is a widely distributed program, used

```
void dumpnode(node *curr, char indent[80])
{  char nextindent[80] ;

   fprintf(dump, " n%s[n%i p%i c%i s%i d%i pat
                  %i pah%i pax%li pau%li dp%+21i", indent,
      curr->lbl, curr->p->lbl, curr->c->lbl, curr->s->lbl,
      curr->d, curr->pa->t->lbl, curr->pa->h->lbl,
      curr->pa->x, curr->pa->u, curr->dp) ;
   strcpy(nextindent, indent) ;
   strcat(nextindent, "  ") ;
   if (curr->c != NULL) dumpnode(curr->c, nextindent) ;
   if (curr->s != NULL) dumpnode(curr->s, indent) ;
   return ;
}

void dumparc(arc *curr)
{  while(curr != NULL)
   {  fprintf(dump, " n  [t%i h%i x%li u%li r%+21i",
         curr->t->lbl, curr->h->lbl,
         curr->x, curr->u, curr->r) ;
      curr = curr->next ;
   }
   return ;
}

void dumpale(ale *curr)
{  while(curr != NULL)
   {  fprintf(dump, " n  [t%i h%i x%li u%li r%+21i",
         curr->a->t->lbl, curr->a->h->lbl,
         curr->a->x, curr->a->u, curr->a->r) ;
      curr = curr->next ;
   }
   return ;

}
```

Figure 4. Data Snapshot Functions.

in over 100 projects for 15 years, with no reported cases of incorrect solutions. No discrepancies were found.

V. TUNING PARAMETER STUDY

A primary research effort examined how characteristics or measures of the network problem's topology influenced the effects of program tuning parameters. The ultimate goal of this research was to be able to calculate some simple measures of the problem and then use these measures to select the tuning parameters that could solve the problem in the fastest time.

This line of research eventually revealed that SNET is influenced only minimally by the choice of tuning parameters, let alone the characteristics of the network problem. Nonetheless, the author believes that the process used in the tuning parameter study could, with other problem measures or program tuning parameters, produce more interesting results in future experiments. This chapter describes that general research process.

A. PROBLEM CHARACTERISTICS

In selecting measures of the network problem to examine, one fact must remain paramount: if the calculations needed to derive the measures are too intensive, more time may be spent calculating them than would be required to solve the problem without setting the tuning parameters. Thus the measures must be simple to calculate. Preferably, they should be derivable as the problem data is being read into the program.

Three characteristics were studied: connectivity, exogenaity, and capacitance. Connectivity (conn) measures the degree of connection between a network's nodes. It is defined as the number of arcs divided by the number of nodes squared. Its value can range from (essentially) zero to one. The connectivity of a network that is sparsely connected, for example a spanning tree, would be close to zero. A network that is completely connected, with an arc between every pair of nodes, would receive a value of one.

Exogenaity (exog) measures the degree of exogenous (external) communication a network has with its environment. It is defined as the number of exogenous nodes (supply or sink nodes) divided by the total number of nodes. Its value can range from zero to one. The exogenaity of a network that has few supply and sink nodes would be close to zero. A transportation or assignment network, where every node is exogenous, would receive a value of one.

Capacitance (cap) is a coarse measure of how much additional flow a network can tolerate. It is defined as the total flow from all supply nodes divided by the sum of the arc's capacities. Its range is also from zero to one, although in practical networks its value is likely to be less than one half. A network with a very low capacitance value can probably handle more flow, while one with a moderate or high level will have more difficulty accommodating increased flow.

B. TUNING PARAMETERS

Tuning parameters are meant to guide the behavior of a program, hopefully leading to improved performance. Possible tuning parameters applicable to network programs include the number of arcs examined during each collection, the size of the candidate queue, the number of candidates examined before incoming arc selection and the value of Big M. Although several tuning parameters may be set within SNET, this research examined only one, percan.

Recall that percan, the percentage of out of basis candidates, is used to set maxnbrcan, the maximum number of favorable candidates encountered and considered for inclusion in the candidate queue during each collection, as follows:

maxnbrcan = percan x (number of real arcs - number of nodes)

maxnbrcan limits the number of out of basis arcs examined during each collection by collect_can.

C. EXAMPLE GENERATION

An external driver program, DSSTEST (Appendix D), was used to generate the examples for the study. DSSTEST uses three nested loops to vary the values of connectivity, exogenaity, and capacitance. Each variable assumed five different values, for a combination of 125 three-tuples. Each tuple was fed, via a translation routine, to NETGEN which generates a random network problem whose characteristics are equal to the tuple values. A fourth nested loop increments percan and then calls SNET to solve the network problem. During the experiment percan assumed 19 different values. The time required for each call to SNET is recorded by DSSTEST, which then writes the (con, exog, cap, percan, solution time) five-tuple to an external data file. DSSTEST generated and recorded 2375 five-tuple examples.

Before analysis, the example set was reduced by selecting the fastest example from each of the 125 problems and then deleting the solution time from each example. Thus, the example set used for analysis (Appendix E) consisted of 125 four-tuples; each con-

taining the three-tuple that represents the characteristics of the problem and the tuning parameter that produced the fastest solution for that problem.

It should be noted that the differences in solution times for each NETGEN problem were usually minor once percan exceeded a low (roughly 30 percent) threshold.

D. ANALYSIS

The examples were then analyzed using two quite different approaches: inductive learning, and multivariate linear regression analysis.

1. Inductive Learning

The inductive reasoning process starts with specific examples and attempts to develop general rules. Hopefully, these generalizing rules can replicate the knowledge contained in the examples in a more compact form.

Quinlan's ID3 induction algorithm [Ref. 12: pp. 167-173] was used to analyze the reduced example set. The ID3 algorithm builds rules in the form of a compact decision tree. At each node, ID3 examines the examples available to it. Using them, it calculates how well each of the unused example characteristics predicts the parameter value. The characteristic that best performs this function is marked as *used* and then serves as the criteria for dividing the examples among the node's children. Although each characteristic value can generate a child, values are usually aggregated to generate the fewest children possible. This division process starts at the tree's root node and continues, recursively, until all the examples assigned to a node have the same parameter value.

The decision tree generated by ID3 (Appendix F) is not significantly more compact than the original example set. It has 101 terminal nodes. The example set contained 125 examples. Thus ID3's induction could not provide general rules given this example set.

2. Regression Analysis

Multivariate linear regression provided more fruitful results. Stepwise regression was applied to the same reduced example set. Analysis of the output (Appendix G) revealed that exogenaity was the only characteristic that had any statistically significant impact upon percan. Though unquestionably important, exogenaity could only explain about 50 percent of the variance in the value of the percan.

Thus neither analysis technique could provide an adequate method of setting the percan parameter given the problem's characteristics.

E. DYNAMIC TUNING

Although this research concentrated on static (unchanging) parameters and characteristics, a similar methodology could be employed in the analysis of dynamic parameters and characteristics. A study of dynamically varying tuning parameters during program execution, based upon dynamic problem characteristics, may provide clues for solving problems faster.

Each of the tuning parameters given above can be altered dynamically. However dynamic problem characteristics are more difficult to derive. Possible candidates for future study include the slope of the objective function or the percentage of flow on artificial arcs. This is an area that deserves further study.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

SNET is approximately twice as fast as a primal simplex network solver written in FORTRAN. SNET's algorithm is not new in the world of network solvers, only its method of storing and retrieving data. As discussed in Chapter 3, its pointer linked data structures allow information to be retrieved and manipulated with minimal computational effort. On the 40 problem NETGEN test set, assigning equal weight to each problem, SNET is 127 percent faster than GNET, a primal simplex network solver written in FORTRAN. The test environment used and specific test times are provided in Appendices A and B.

SNET is relatively insensitive to the tuning parameter examined. As noted in Chapter 5, SNET's tuning parameter, percan, was tested over 19 values for 125 different random problems. For most problems, once percan exceeded the 30 percent threshold, its effect on solution time was negligible.

When new programming features become available for incorporation into a solver, fundamental assumptions may no longer be valid and, therefore, should be reexamined. Before the use of true pointer data structures, optimizers spoke of *pivots* being cheap and *price-outs* being expensive. As the profiler reveals, this is not true when pointer based structures maintain the data. Many examples of this maxim were encountered during SNET's development.

B. RECOMMENDATIONS

The author has two recommendations for future research: one concerning SNET and the other dealing with general network problems.

First, SNET--though fast--is not a mature solver. It could benefit from improvements in many areas. Most serious though, is the number of pivots that SNET requires to solve a problem. A better method of selecting incoming arcs is needed to reduce the number of pivots. Although SNET is faster than GNET, it also requires more pivots. Its candidate selection method is simple and fast, but a better pivot sequence, though more expensive to acquire, could significantly reduce total solution time.

Three general suggestions are offered for improving the pivot sequence. Each of these areas could serve as a focus for future research:

- Maintain the out of basis arcs in parallel linked lists, with list membership determined by a common head or tail node. This data structure would improve the efficiency of search routines and allow arcs that enter or leave a specific node to be quickly located.
- Research the effects of other sorting and searching methods during candidate collection (bringing arcs into the candidate queue) and candidate selection (choosing an arc from the candidate queue to enter the basis).
- Examine tuning parameter selection as a function of problem characteristics, both static and dynamic.

Second, since true pointer based structures improved solver efficiency in solving min-cost network problems, perhaps they can improve the efficiency of shortest path, min-cost spanning tree, traversals, and other network solvers.

APPENDIX A. TEST ENVIRONMENT

SNET and GNET were tested under the same environment. All tests were run on a NeXT Computer, model number N9001, with version 1.0 of the NeXTSTEP operating system and a Motorola 68030 25MHz CPU.

The NETGEN problems were stored on an optical disk and then copied, one at a time, to a 330 Megabyte hard disk. Each solver read the problem from the hard drive. The Next's 16 Megabytes of main memory was more than sufficient for each of the problems and no paging was required.

The combined problem read in and solution time of each solver was recorded for comparison purposes. Time used was measured by the C clock function.

GNET is written in FORTRAN 77 and was compiled using the Absoft FORTRAN compiler, NeXT version 2.0, with the optimizing flag turned on. GNET's tuning parameters were set as recommended by its developers.

SNET is a C program. It was compiled with the NeXT version of the GNU C compiler developed by the Free Software Foundation. SNET was also compiled with the optimizing flag turned on. Its tuning parameter, percan, was not set via the command line argument. Therefore, by default, maxnbrcan was set to MAXCAN (20,000).

APPENDIX B. TIME TRIALS: SNET VS. GNET

netgen	snet	gnet	fraction
1	2.53	4.92	0.51
2	2.80	5.34	0.52
3	3.22	6.75	0.48
4	3.50	7.30	0.48
5	3.86	9.22	0.42
6	4.66	10.58	0.44
7	6.25	14.67	0.43
8	7.05	16.66	0.42
9	7.56	18.48	0.41
10	8.09	19.80	0.41
11	3.23	6.25	0.52
12	3.56	8.52	0.42
13	4.61	10.39	0.44
14	5.45	12.78	0.43
15	5.56	15.20	0.37
16	2.25	4.72	0.48
17	3.03	7.88	0.38
18	2.36	4.86	0.49
19	3.11	7.59	0.41
20	2.39	5.09	0.47
21	3.38	8.69	0.39
22	2.34	5.00	0.47
23	3.22	8.58	0.38
24	2.42	4.95	0.49
25	3.64	8.27	0.44
26	1.95	4.69	0.42
27	3.22	8.09	0.40
28	4.30	10.70	0.40
29	5.33	11.88	0.45
30	5.80	14.52	0.40
31	6.25	15.77	0.40
32	6.59	16.70	0.39
33	6.69	16.61	0.40
34	7.52	19.05	0.39
35	8.61	20.78	0.41
36	54.34	123.06	0.44
37	55.42	105.75	0.52
38	56.20	125.67	0.45
39	41.55	81.41	0.51
40	36.95	82.39	0.45
mean difference in speed			0.44

APPENDIX C. PROGRAM LISITING: TESTEXEC

```
/* Time network solvers on standard NETGEN problems  */

/* Header file */
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Print an error message to the screen and stop the program   */
void halt(char message[80])
{  printf(" nHalt invoked, error in %s process. n", message) ;
   exit( 1 ) ;
}

/* reverse: reverse string s in place */
void reverse(char s[])
{  int c, i, j ;

   for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
   {  c = s[i] ;
      s[i] = s[j] ;
      s[j] = c ;
   }
}

/* itoa:  convert n to characters in s */
char *itoa(int n)
{  int i = 0, sign ;
   char s[25] ;

   if ((sign = n) < 0) n = -n ;
   do  {  s[i++] = n % 10 + '0' ;
       }  while ((n /= 10) > 0) ;
   if ( sign < 0) s[i++] = '-' ;
      s[i] = ' 0' ;
   reverse(s) ;
   return(s) ;
}

void main()
{  int      i, first, last ;
   long     start,
            end ;
   float    deltime1, deltime2, fraction, sum = 0  ;
   char     infilename[80],
            outfilename[80],
            code[25],
            command[80];
   FILE     *infile,
            *timefile ;
```

```
/* Open timing file */
if ((timefile = fopen("raw.time", "a")) == NULL)
   halt("opening timing file") ;

/* Print timing file header */
fprintf(timefile, "netgen     snet        gnet     fraction n") ;

/* Main loop */
/* NOTE: be sure loop values are correct before final tests */
first = 1 ;
last = 40 ;
for (i = first; i <= last; i++)
{   /* Get filename for optical disk */
    strcpy(code, itoa(i)) ;
    strcpy(infilename, "/NetGenDisk/problems/net") ;
    if (i < 10) strcat(infilename, "0") ;
    strcat(infilename, code) ;

    /* Set local filename */
    strcpy(outfilename, "net") ;
    if (i < 10) strcat(outfilename, "0") ;
    strcat(outfilename, code) ;

    /* Inform console */
    printf(" n**************** PROBLEM NET%i **************** n", i) ;

    /* Copy file from optical disk to hard disk */
    strcpy(command, "cp ") ;
    strcat(command, infilename) ;
    strcat(command, " ") ;
    strcat(command, outfilename) ;
    system(command) ;

    /* Time pointer version of snet */
    strcpy(command, "time pointer ") ;
    strcat(command, outfilename) ;
    start = clock() ;
    system(command) ;
    end = clock() ;
    deltime1 = ((float)(end-start) / CLK_TCK) ;
    fprintf(timefile," %8.2f  ",deltime1) ;

    /* Time GNET */
    printf(" n") ;
    rename(outfilename, "net") ;
    strcpy(command, "time gnet ") ;
    start = clock() ;
    system(command) ;
    end = clock() ;
    deltime2 = ((float)(end-start) / CLK_TCK) ;
    fprintf(timefile," %8.2f ",deltime2) ;

    /* Calculate snet as a fraction of GNET */
    fraction = deltime1 / deltime2 ;
    fprintf(timefile," %8.2f n",fraction) ;
    sum += fraction ;
```

```
        /* Delete file from hard disk */
        remove("net") ;
    }
     sum = sum / ((last - first) + 1) ;
     fprintf(timefile," n     mean difference in speed %8.2f n",sum) ;
     fclose(timefile) ;
}
```

APPENDIX D. PROGRAM LISITING: DSSTEST

```
/* Time random network problems and program parameters */

/* Header file */
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Print an error message to the screen and stop the program   */
void halt(char message[80])
{  printf(" nHalt invoked, error in %s process. n", message) ;
   exit( 1 ) ;
}

/* reverse: reverse string s in place */
void reverse(char s[])
{  int c, i, j ;

   for (i = 0, j = strlen(s) - 1; i < j; i++, j--)
   {  c = s[i] ;
      s[i] = s[j] ;
      s[j] = c ;
   }
}

/* itoa:  convert n to characters in s */
char *itoa(int n)
{  int i = 0, sign ;
   char s[25] ;

   if ((sign = n) < 0) n = -n ;
   do  {  s[i++] = n % 10 + '0' ;
       }  while ((n /= 10) > 0) ;
   if ( sign < 0) s[i++] = '-' ;
      s[i] = ' 0' ;
   reverse(s) ;
   return(s) ;
}

void main()
{  int      nbr_nodes = 200,
           min_arc_cost = 1,
           max_arc_cost = 100,
           nbr_arcs,
           nbr_exog_nodes,
           nbr_sup_nodes,
           nbr_dem_nodes,
           max_up_bnd,
           min_up_bnd,
           parameter,
```

```
            j ;
long        start,
            end ;
float       tot_supply = 100000.0,
            frac_arcs_cap = 0.75,
            epsilon = 0.0003,
            sum_arc_cap,
            deltim,
            con,
            exog,
            cap ;
char        infilename[25],
            code[25],
            dummy[80],
            command[80],
            scale[500],
            rand_nbr[] = "39962782",
            parstr[25] ;
FILE        *infile,
            *timefile ;

/* Open timing file */
if ((timefile = fopen("cantimes", "w")) == NULL)
   halt("opening timing file") ;

/* Main loop */
/* NOTE: be sure loop values are correct before final test */

for (con = 0.05; con <= 0.85 + epsilon; con +=0.20)
{  for (exog = 0.10; exog <= 0.90 + epsilon; exog +=0.20)
   {  for (cap = 0.10; cap <= 0.50 + epsilon; cap +=0.10)
      {  nbr_arcs = nbr_nodes * nbr_nodes * con ;
         nbr_exog_nodes = nbr_nodes * exog ;
         nbr_sup_nodes = nbr_exog_nodes * 0.20 ;
         nbr_dem_nodes = nbr_exog_nodes * 0.80 ;
         sum_arc_cap = tot_supply / cap ;
         max_up_bnd = sum_arc_cap / nbr_arcs ;
         min_up_bnd = 0.25 * max_up_bnd ;

         strcpy(code, itoa(con * 100)) ;
         strcat(code, itoa(exog *100)) ;
         strcat(code, itoa(cap * 100)) ;
         strcpy(infilename, "ni") ;
         strcat(infilename, code) ;

         if ((infile = fopen(infilename, "w")) == NULL)
            halt("opening NETGEN 'in' file") ;
         fprintf(infile, "%s  n", rand_nbr) ;
         fprintf(infile,
                 "%5i%5i%5i%5i%5i%5i%10li%5i%5i%5.1f%5.1f%10i%10i n",
                 nbr_nodes,
                 nbr_sup_nodes,
                 nbr_dem_nodes,
                 nbr_arcs,
                 min_arc_cost,
                 max_arc_cost,
```

```
                    (long)tot_supply,
                    0,
                    0,
                    5.0,
                    100.0 * frac_arcs_cap,
                    min_up_bnd,
                    max_up_bnd ) ;
            fclose(infile) ;
            rename(infilename, "netin") ;

            printf("generating problem for code %s n", code) ;
            system("netgen") ;
            printf("solving problem with captran n") ;
            fprintf(timefile, "code: %s n", code) ;

            /* NOTE: correct loop values before final test */
            for (parameter = 5; parameter <= 95; parameter += 5)
            {  strcpy(parstr, itoa(parameter)) ;
               strcpy(command, "captran net ") ;
               strcat(command, parstr) ;
               strcat(command, " >outfile") ;

               start = clock() ;
               system(command) ;
               end = clock() ;
               deltim = ((float)(end-start) / CLK_TCK) ;
               scale[0] = ' 0' ;
               for (j = 1; j <= (int)(deltim); j++)
                  strcat(scale, "*") ;
               fprintf(timefile,"%4i %4i %4i %4i %6.2f %s n",
                        (int)(con*100), (int)(exog*100), (int)(cap*100),
                        parameter, deltim, scale) ;

            }
            fprintf(timefile, " n n") ;
            printf(" n") ;

            remove("netin") ;
            remove("net") ;
            remove("netrep") ;
         }
      }
   }
    fclose(timefile) ;
}
```

APPENDIX E. EXAMPLE SET (REDUCED)

This reduced example set consists of 125 four-tuples. They are the Connectivity, Exogenaity, and Capacitance network problem characteristics and the best (fastest) tuning parameter choice for 125 random problems. The example set is used as sample data for Induction and Regression Analysis.

Problem Characteristics			Best Parameter
Con	Exog	Cap	Percan
5	10	10	25
5	10	20	55
5	10	30	30
5	10	40	25
5	10	50	35
5	30	10	30
5	30	20	45
5	30	30	15
5	30	40	50
5	30	50	50
5	50	10	80
5	50	20	30
5	50	30	35
5	50	40	30
5	50	50	25
5	70	10	35
5	70	20	50
5	70	30	35
5	70	40	75
5	70	50	55
5	90	10	55
5	90	20	55
5	90	30	55
5	90	40	75
5	90	50	55
25	10	10	20
25	10	20	30
25	10	30	10
25	10	40	10
25	10	50	20
25	30	10	35
25	30	20	35
25	30	30	25
25	30	40	15
25	30	50	50
25	50	10	35
25	50	20	45
25	50	30	45
25	50	40	35
25	50	50	80

25	70	10	45
25	70	20	45
25	70	30	55
25	70	40	85
25	70	50	45
25	90	10	40
25	90	20	35
25	90	30	30
25	90	40	35
25	90	50	50
45	10	10	5
45	10	20	15
45	10	30	10
45	10	40	10
45	10	50	15
45	30	10	25
45	30	20	20
45	30	30	25
45	30	40	85
45	30	50	30
45	50	10	40
45	50	20	40
45	50	30	40
45	50	40	40
45	50	50	60
45	70	10	80
45	70	20	30
45	70	30	60
45	70	40	55
45	70	50	65
45	90	10	55
45	90	20	60
45	90	30	55
45	90	40	45
45	90	50	45
65	10	10	25
65	10	20	15
65	10	30	5
65	10	40	20
65	10	50	20
65	30	10	10
65	30	20	30
65	30	30	25
65	30	40	15
65	30	50	5
65	50	10	45
65	50	20	30
65	50	30	40
65	50	40	25
65	50	50	30
65	70	10	60
65	70	20	75
65	70	30	60
65	70	40	60
65	70	50	50
65	90	10	80

65	90	20	35
65	90	30	45
65	90	40	80
65	90	50	90
85	10	10	10
85	10	20	10
85	10	30	10
85	10	40	10
85	10	50	15
85	30	10	25
85	30	20	15
85	30	30	5
85	30	40	5
85	30	50	5
85	50	10	75
85	50	20	30
85	50	30	40
85	50	40	30
85	50	50	35
85	70	10	60
85	70	20	55
85	70	30	50
85	70	40	50
85	70	50	60
85	90	10	60
85	90	20	60
85	90	30	80
85	90	40	95
85	90	50	80

APPENDIX F. INDUCTIVE DECISION TREE

Decision tree for reduced example set

```
 1: exog??
 2:     <20.00: con??
 3:       <15.00: cap??
 4:         <45.00: cap??
 5:           <25.00: cap??
 6:             <15.00: ----------------------------------25
 7:             >15.00: ----------------------------------55
 8:           >25.00: cap??
 9:             <35.00: ----------------------------------30
10:             >35.00: ----------------------------------25
11:         >45.00: ------------------------------------35
12:       >15.00: cap??
13:         <45.00: cap??
14:           <25.00: con??
15:             <75.00: cap??
16:               <15.00: con??
17:                 <35.00: ----------------------------20
18:                 >35.00: con??
19:                   <55.00: ----------------------------5
20:                   >55.00: ----------------------------25
21:               >15.00: con??
22:                 <35.00: ----------------------------30
23:                 >35.00: ----------------------------15
24:             >75.00: ----------------------------------10
25:           >25.00: con??
26:             <55.00: ----------------------------------10
27:             >55.00: con??
28:               <75.00: cap??
29:                 <35.00: ----------------------------5
30:                 >35.00: ----------------------------20
31:               >75.00: ----------------------------------10
32:         >45.00: con??
33:           <35.00: ----------------------------------20
34:           >35.00: con??
35:             <55.00: ----------------------------------15
36:             >55.00: con??
37:               <75.00: ----------------------------20
38:               >75.00: ----------------------------15
39:     >20.00: exog??
40:       <60.00: exog??
41:         <40.00: cap??
42:           <35.00: con??
43:             <15.00: cap??
44:               <15.00: ----------------------------30
45:               >15.00: cap??
46:                 <25.00: ----------------------------45
47:                 >25.00: ----------------------------15
48:             >15.00: con??
```

```
49:                 <55.00: con??
50:                   <35.00: cap??
51:                     <25.00: ----------35
52:                     >25.00: ----------25
53:                   >35.00: cap??
54:                     <15.00: ----------25
55:                     >15.00: cap??
56:                       <25.00: ----------20
57:                       >25.00: ----------25
58:                 >55.00: con??
59:                   <75.00: cap??
60:                     <15.00: ----------10
61:                     >15.00: cap??
62:                       <25.00: ----------30
63:                       >25.00: ----------25
64:                   >75.00: cap??
65:                     <15.00: ----------25
66:                     >15.00: cap??
67:                       <25.00: ----------15
68:                       >25.00: ----------5
69:             >35.00: con??
70:               <35.00: con??
71:                 <15.00: ----------50
72:                 >15.00: cap??
73:                   <45.00: ----------15
74:                   >45.00: ----------50
75:               >35.00: con??
76:                 <55.00: cap??
77:                   <45.00: ----------85
78:                   >45.00: ----------30
79:                 >55.00: con??
80:                   <75.00: cap??
81:                     <45.00: ----------15
82:                     >45.00: ----------5
83:                   >75.00: ----------5
84:           >40.00: con??
85:             <35.00: cap??
86:               <45.00: con??
87:                 <15.00: cap??
88:                   <15.00: ----------80
89:                   >15.00: cap??
90:                     <25.00: ----------30
91:                     >25.00: cap??
92:                       <35.00: ----------35
93:                       >35.00: ----------30
94:                 >15.00: cap??
95:                   <15.00: ----------35
96:                   >15.00: cap??
97:                     <35.00: ----------45
98:                     >35.00: ----------35
99:               >45.00: con??
100:                <15.00: ----------25
101:                >15.00: ----------80
102:              >35.00: con??
103:                <55.00: cap??
104:                  <45.00: ----------40
```

```
105:                >45.00: ----------60
106:              >55.00: cap??
107:                <15.00: con??
108:                  <75.00: ----------45
109:                  >75.00: ----------75
110:                >15.00: cap??
111:                  <35.00: cap??
112:                    <25.00: ----------30
113:                    >25.00: ----------40
114:                  >35.00: con??
115:                    <75.00: cap??
116:                      <45.00: ----------25
117:                      >45.00: ----------30
118:                    >75.00: cap??
119:                      <45.00: ----------30
120:                      >45.00: ----------35
121:          >60.00: con??
122:            <35.00: con??
123:              <15.00: cap??
124:                <35.00: exog??
125:                  <80.00: cap??
126:                    <15.00: ----------35
127:                    >15.00: cap??
128:                      <25.00: ----------50
129:                      >25.00: ----------35
130:                  >80.00: ----------55
131:                >35.00: cap??
132:                  <45.00: ----------75
133:                  >45.00: ----------55
134:              >15.00: exog??
135:                <80.00: cap??
136:                  <25.00: ----------45
137:                  >25.00: cap??
138:                    <35.00: ----------55
139:                    >35.00: cap??
140:                      <45.00: ----------85
141:                      >45.00: ----------45
142:                >80.00: cap??
143:                  <15.00: ----------40
144:                  >15.00: cap??
145:                    <45.00: cap??
146:                      <25.00: ----------35
147:                      >25.00: cap??
148:                        <35.00: ----------30
149:                        >35.00: ----------35
150:                    >45.00: ----------50
151:            >35.00: exog??
152:              <80.00: con??
153:                <55.00: cap??
154:                  <25.00: cap??
155:                    <15.00: ----------80
156:                    >15.00: ----------30
157:                  >25.00: cap??
158:                    <35.00: ----------60
159:                    >35.00: cap??
160:                      <45.00: ----------55
```

```
161:                    >45.00: ----------65
162:                >55.00: cap??
163:                  <25.00: cap??
164:                    <15.00: ----------60
165:                    >15.00: con??
166:                      <75.00: ----------75
167:                      >75.00: ----------55
168:                  >25.00: con??
169:                    <75.00: cap??
170:                      <45.00: ----------60
171:                      >45.00: ----------50
172:                    >75.00: cap??
173:                      <45.00: ----------50
174:                      >45.00: ----------60
175:              >80.00: cap??
176:                <25.00: cap??
177:                  <15.00: con??
178:                    <55.00: ----------55
179:                    >55.00: con??
180:                      <75.00: ----------80
181:                      >75.00: ----------60
182:                  >15.00: con??
183:                    <55.00: ----------60
184:                    >55.00: con??
185:                      <75.00: ----------35
186:                      >75.00: ----------60
187:                >25.00: con??
188:                  <55.00: cap??
189:                    <35.00: ----------55
190:                    >35.00: ----------45
191:                  >55.00: con??
192:                    <75.00: cap??
193:                      <35.00: ----------45
194:                      >35.00: cap??
195:                        <45.00: ----------80
196:                        >45.00: ----------90
197:                    >75.00: cap??
198:                      <35.00: ----------80
199:                      >35.00: cap??
200:                        <45.00: ----------95
201:                        >45.00: ----------80
```

APPENDIX G. REGRESSION ANALYSIS

STEPWISE REGRESSION OF percan ON 3 PREDICTORS, WITH N = 125

STEP	1
CONSTANT	13.02
exog	0.542
T-RATIO	10.84
S	15.8
R-SQ	48.86

LINEAR REGRESSION OF percan ON 1 PREDICTOR exog

The regression equation is
percan = 13.0 + 0.542 exog

Predictor	Coef	Stdev	t-ratio	p
Constant	13.020	2.872	4.53	0.000
exog	0.54200	0.05000	10.84	0.000

s = 15.81 R-sq = 48.9% R-sq(adj) = 48.4%

Analysis of Variance

SOURCE	DF	SS	MS	F	p
Regression	1	29376	29376	117.52	0.000
Error	123	30747	250		
Total	124	60123			

Unusual Observations

Obs.	exog	percan	Fit	Stdev.Fit	Residual	St.Resid
2	10.0	55.00	18.44	2.45	36.56	2.34R
11	50.0	80.00	40.12	1.41	39.88	2.53R
40	50.0	80.00	40.12	1.41	39.88	2.53R
44	70.0	85.00	50.96	1.73	34.04	2.17R
48	90.0	30.00	61.80	2.45	-31.80	-2.04R
59	30.0	85.00	29.28	1.73	55.72	3.55R
111	50.0	75.00	40.12	1.41	34.88	2.21R
124	90.0	95.00	61.80	2.45	33.20	2.13R

LINEAR REGRESSION OF percan ON 3 PREDICTORS, WITH N = 125

The regression equation is
percan = 13.5 - 0.0500 con + 0.542 exog + 0.058 cap

Predictor	Coef	Stdev	t-ratio	p
Constant	13.530	4.736	2.86	0.005
con	-0.05000	0.05013	-1.00	0.321
exog	0.54200	0.05013	10.81	0.000

cap	0.0580	0.1003	0.58	0.564

s = 15.85 R-sq = 49.4% R-sq(adj) = 48.2%

Analysis of Variance

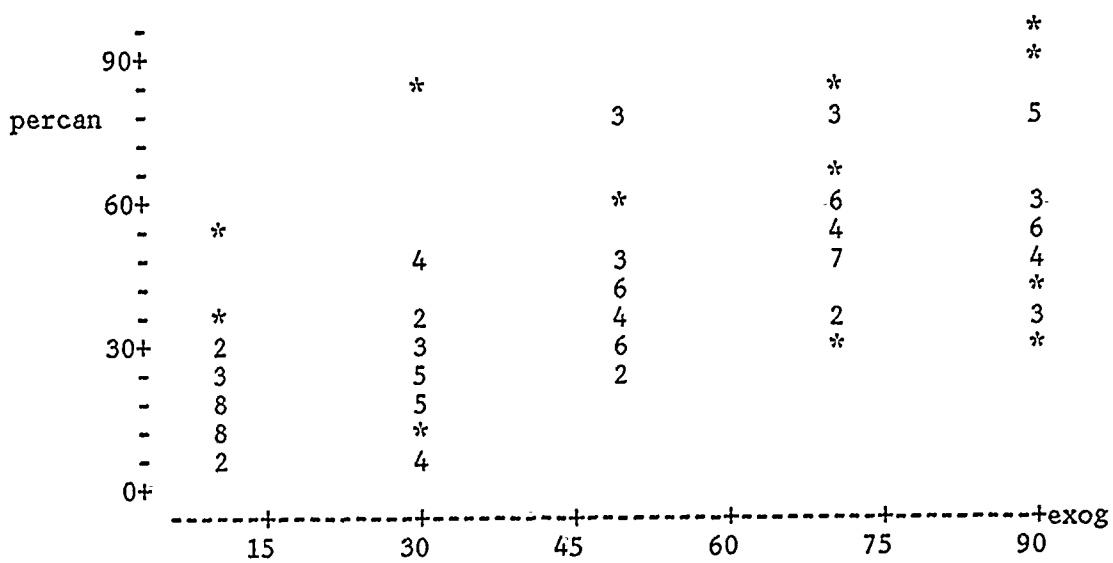
SOURCE	DF	SS	MS	F	p
Regression	3	29710.5	9903.5	39.40	0.000
Error	121	30412.7	251.3		
Total	124	60123.2			

SOURCE	DF	SEQ SS
con	1	250.0
exog	1	29376.4
cap	1	84.1

Unusual Observations

Obs.	con	percan	Fit	Stdev.Fit	Residual	St.Resid
2	5.0	55.00	19.86	3.33	35.14	2.27R
11	5.0	80.00	40.96	3.17	39.04	2.51R
40	25.0	80.00	42.28	2.65	37.72	2.41R
44	25.0	85.00	52.54	2.24	32.46	2.07R
48	25.0	30.00	62.80	2.65	-32.80	-2.10R
59	45.0	85.00	29.86	2.01	55.14	3.51R
111	85.0	75.00	36.96	3.17	38.04	2.45R
124	85.0	95.00	60.38	3.33	34.62	2.23R

PLOT 'percan' vs. 'exog'



LIST OF REFERENCES

1. Bazaraa, M. S., and Jarvis, J. J., *Linear Programming and Network Flows*, John Wiley & Sons, Inc., 1977.

2. Bradley, G. H., Brown, G. G., and Graves, G. W., "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, v.24 n.1, pp. 1-34, September 1977.

3. Grigoriadis, M. D., and Hsu, T., *The Minimum Cost Network Flow Subroutines (RNET documentation)*, Department of Computer Science, Rutgers University, November 1980.

4. MIT Operations Research Center, Report 055-76, *Implementing Primal-Dual Network Flow Algorithm*, Aashtiani, H. A., and Magnanti, T., June 1976.

5. Bertsekas, D. R., and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems," *Operations Research*, v.36 n.1, pp. 93-114, February 1988.

6. Dantzig, G. *Linear Programming and Extensions*, Princeton University Press, 1963.

7. Rosenthal, R.E., unpublished notes: "The Capacitated Transshipment Problem," Naval Postgraduate School, Monterey, CA, 1990.

8. Harary, F., *Graph Theory*, Addison-Wesley Publishing Co., 1969.

9. Lustig, I.J., "The Influence of Computer Language on Computational Comparisons: An Example from Network Optimization," *ORSA Journal on Computing*, v.2 n.2, pp. 152-161, Spring 1990.

10. Van Wyk, C.J., *Data Structures and C Programs*, Addison-Wesley Publishing Co., 1988.

11. Klingman, D., Napier, A. and Stutz, J., "NETGEN: A Program for Generating Large-scale Assignment, Transportation, and Minimum Cost Flow Network Problems," *Management Science*, v.20 n.5, pp. 814-821, January 1974.

12. Kid, A. L., *Knowledge Acquisition for Expert Systems*, Plenum Press, 1987.

BIBLIOGRAPHY

Aho, A.V., Hopcraft, J. E., and Ullman, J.D., *Data Structures and Algorithms*, Addison-Wesley Publishing Co., 1987

Kernighan, B.W., and Ritchie, D.M., *The C Programming Language*, Prentice Hall, 1988

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Director Attn: Mr. E.B. Vandiver III U.S. Army Concepts Analysis Agency Bethesda, MD 20814	1
4.	Commander U.S. Army TRADOC Analysis Command Attn: ATRC Fort Leavenworth, KS 66027-5200	1
5.	USSPACECOM; Flash J3SOX ATTN: CPT S. Rapp Cheyenne Mountain AFB Colorado Springs, CO 80914	1
6.	Deputy Undersecretary of the Army for Operations Research Room 2E261, The Pentagon Washington, D.C. 20310	1
7.	HQDA Office of the Technical Advisor Deputy Chief of Staff for Operations and Plans Attn: DAMO-ZD Room 3A538, The Pentagon Washington, D.C. 20310-0401	1
8.	Director U.S. Army TRADOC Analysis Command - Fort Leavenworth Attn: ATRC-FOQ (Technical Information Center) Fort Leavenworth, KS 66027-5200	1
9.	Professor Gordon H. Bradley Department of Operations Research Naval Postgraduate School, Code OR/BZ Monterey, CA 93943-5000	2

10. LCDR R. Stemp 1
Department of Operations Research
Naval Postgraduate School, Code 30
Monterey, CA 93943-5000

11. CPT Keith D. Solveson 1
Rt. 1
Plum City, WI 54761